Using MaxSAT for Efficient Explanations of Tree Ensembles

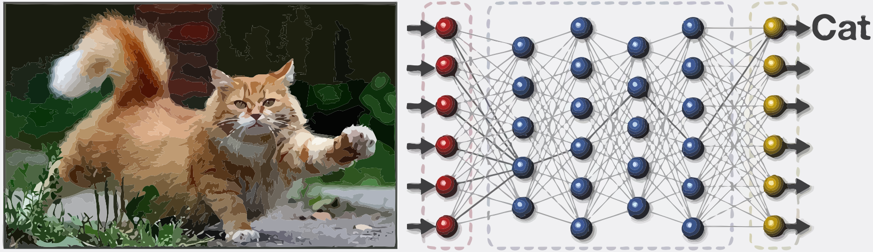
Alexey Ignatiev[1], Yacine Izza[2], Peter J. Stuckey[1], Joao Marques-Silva[3]

February 22 – March 1, 2022 | **AAAI**

[1] Monash University, Melbourne, Australia
[2] University of Toulouse, France
[3] IRIT, CNRS, Toulouse, France

Formal eXplainable AI

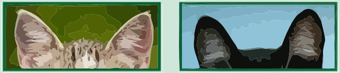
Machine Learning System



This is a cat.

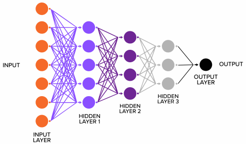
Current Explanation

This is a cat:
- It has fur, whiskers, and claws.
- It has this feature:



XAI Explanation

Why? Status quo...

	A parrot 	Machine learning algorithm 
Learns random phrases	✓	✓
Doesn't understand s**t about what it learns	✓	✓
Occasionally speaks nonsense	✓	✓

classifier $\tau : \mathbb{F} \rightarrow \mathcal{K}$, **instance** $\mathbf{v}$ **s.t.** $\tau(\mathbf{v}) = c$

classifier $\tau : \mathbb{F} \to \mathcal{K}$, **instance** $\mathbf{v}$ **s.t.** $\tau(\mathbf{v}) = \mathbf{c}$

---

abductive explanation $\mathcal{X}$

$$\forall(\mathbf{x} \in \mathbb{F}). \bigwedge_{j \in \mathcal{X}} (x_j = v_j) \rightarrow (\tau(\mathbf{x}) = \mathbf{c})$$

Formal abductive explanations

classifier $\tau : \mathbb{F} \rightarrow \mathcal{K}$, **instance** $\mathbf{v}$ **s.t.** $\tau(\mathbf{v}) = \mathbf{c}$

---

abductive explanation $\mathcal{X}$

"why prediction c?"

$$\forall(\mathbf{x} \in \mathbb{F}). \bigwedge_{j \in \mathcal{X}} (x_j = v_j) \rightarrow (\tau(\mathbf{x}) = \mathbf{c})$$

classifier $\tau : \mathbb{F} \to \mathcal{K}$, **instance** $\mathbf{v}$ **s.t.** $\tau(\mathbf{v}) = \mathbf{c}$

---

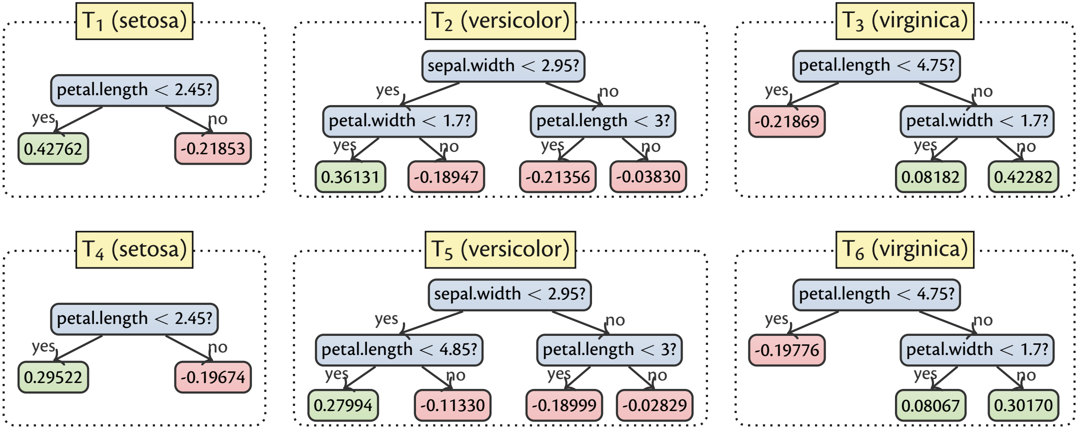
abductive explanation $\mathcal{X}$

"why prediction c?"

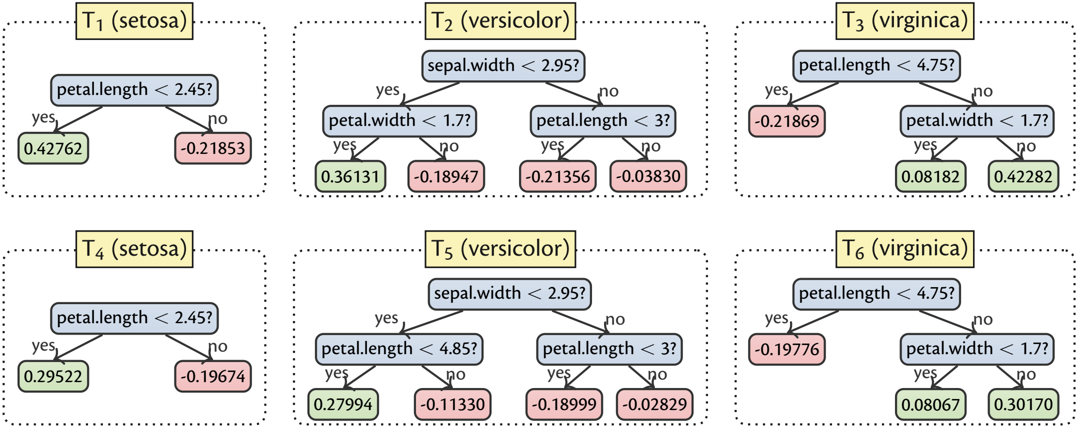
$$\forall(\mathbf{x} \in \mathbb{F}). \bigwedge_{j\in\mathcal{X}}(x_j = v_j) \to (\tau(\mathbf{x}) = \mathbf{c})$$

because of features of $\mathcal{X}$**!**

Abductive explanations for tree ensembles

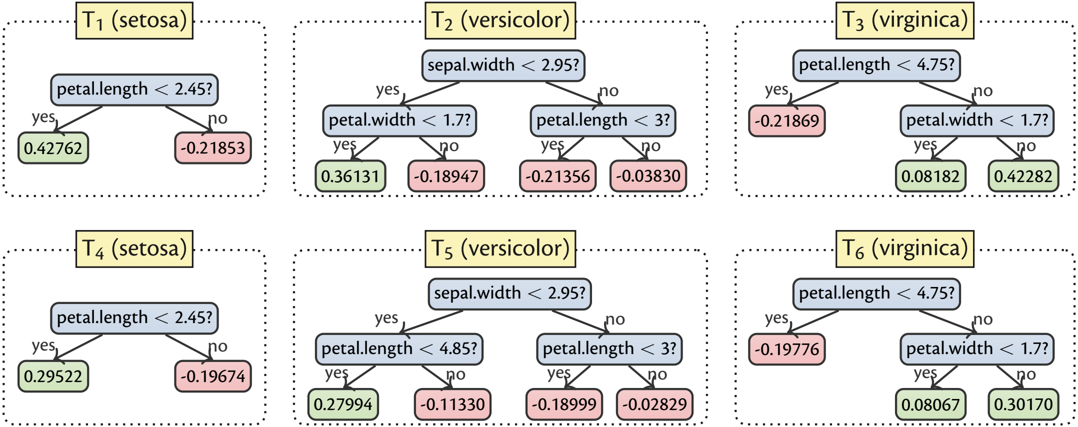


Abductive explanations for tree ensembles



- $w(\mathbf{x}, c) = \sum_{j \in \{0,\ldots,n-1\}} \mathcal{T}_{Kj+c}(\mathbf{x}),\ c \in [K]$
- $\tau(\mathbf{x}) = \arg\max_{c \in [K]} w(\mathbf{x}, c)$

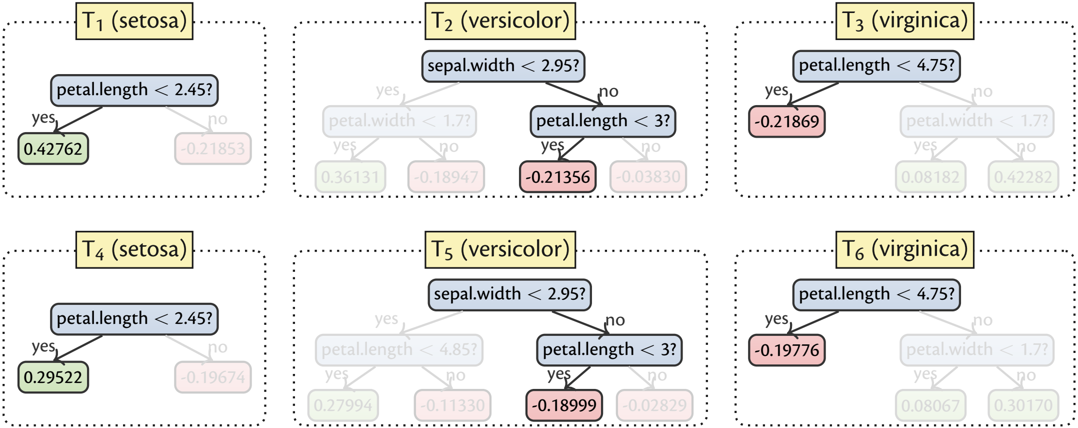
Abductive explanations for tree ensembles



- $w(\mathbf{x}, \mathbf{c}) = \sum_{j \in \{0,\ldots,n-1\}} \mathcal{T}_{Kj+c}(\mathbf{x}), \;\; \mathbf{c} \in [\mathbf{K}]$
- $\tau(\mathbf{x}) = \textbf{arg max}_{c \in [K]} w(\mathbf{x}, \mathbf{c})$

$\forall(\mathbf{x} \in \mathbb{F}). \bigwedge_{j \in \mathcal{X}} (x_j = v_j) \rightarrow (\tau(\mathbf{x}) = \mathbf{c})$

Abductive explanations for tree ensembles

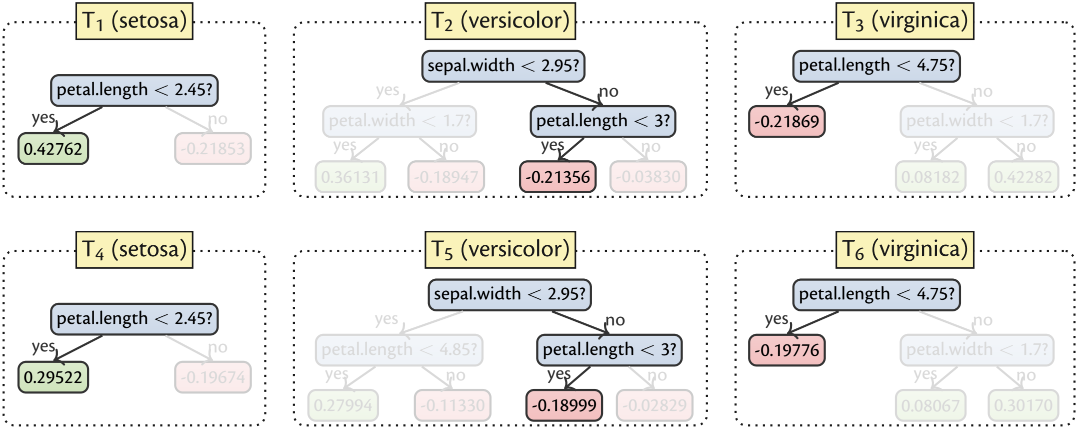


- $w(\mathbf{x}, \mathbf{c}) = \sum_{j \in \{0,\ldots,n-1\}} \mathcal{T}_{Kj+c}(\mathbf{x}),\ \ \mathbf{c} \in [\mathbf{K}]$
- $\tau(\mathbf{x}) = \arg\max_{c \in [K]} w(\mathbf{x}, \mathbf{c})$

$(\text{sepal.length} = 5.1) \land (\text{sepal.width} = 3.5) \land (\text{petal.length} = 1.4) \land (\text{petal.width} = 0.2)$

$\forall (\mathbf{x} \in \mathbb{F}). \bigwedge_{j \in \mathcal{X}} (x_j = v_j) \rightarrow (\tau(\mathbf{x}) = c)$

Abductive explanations for tree ensembles

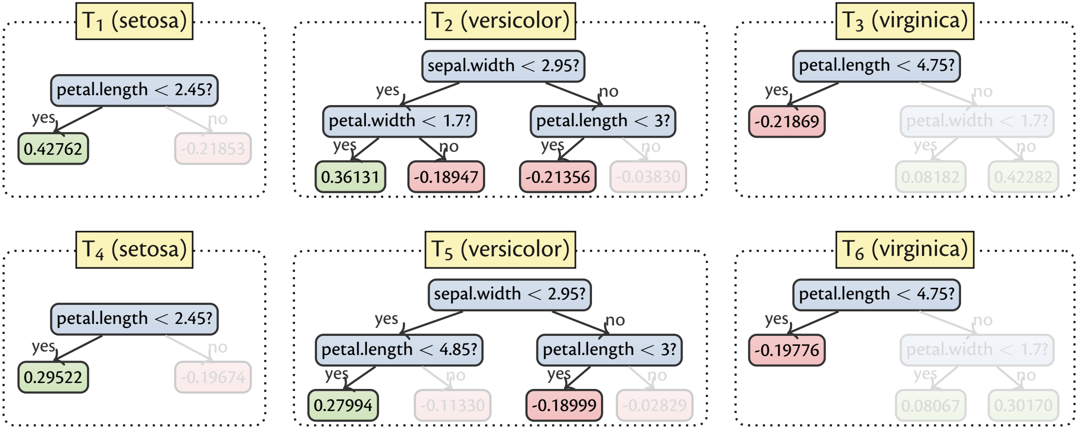


- $w(\mathbf{x}, c) = \sum_{j \in \{0,\dots,n-1\}} \mathcal{T}_{Kj+c}(\mathbf{x}),\ \ c \in [K]$
- $\tau(\mathbf{x}) = \arg\max_{c \in [K]} w(\mathbf{x}, c)$

$(\text{sepal.length} = 5.1) \land (\text{sepal.width} =$ $\mathcal{X} = \{(\text{petal.length} = 1.4)\}$ $(\text{petal.width} = 0.2)$

$\forall(\mathbf{x} \in \mathbb{F}).\bigwedge_{j \in \mathcal{X}} (x_j = v_j) \rightarrow (\tau(\mathbf{x}) = c)$

Abductive explanations for tree ensembles



- $w(\mathbf{x}, c) = \sum_{j \in \{0,\ldots,n-1\}} \mathcal{T}_{Kj+c}(\mathbf{x}), \;\; c \in [K]$
- $\tau(\mathbf{x}) = \arg\max_{c \in [K]} w(\mathbf{x}, c)$

$(\text{sepal.length} = 5.1) \land (\text{sepal.width} =$ $\mathcal{X} = \{(\text{petal.length} = 1.4)\}$ $(\text{petal.width} = 0.2)$

$\forall(\mathbf{x} \in \mathbb{F}). \bigwedge_{j \in \mathcal{X}} (x_j = v_j) \rightarrow (\tau(\mathbf{x}) = c)$

Formal explanations with MaxSAT

Overview of the approach

SMT

- **reach logic, can handle *linear constraints***

Overview of the approach

SMT

- **reach logic, can handle *linear constraints***
- **directly reason about:**

$$\mathcal{H} \wedge \left(\sum_{\iota} \geq \sum_{i}\right)$$

Overview of the approach

SMT

- **reach logic, can handle *linear constraints***
- **directly reason about:**

$$\mathcal{H} \wedge \left(\sum_{\iota} \geq \sum_{i} \right)$$

encode BT operation

Overview of the approach

SMT

- reach logic, can handle *linear constraints*
- directly reason about:

$$\mathcal{H} \wedge \left(\sum_{\iota} \geq \sum_{i}\right)$$

encode BT operation

enforce class c_ι instead of c_i

insane decimal point precision

Overview of the approach

SMT

- **reach logic, can handle *linear constraints***
- **directly reason about:**

$$\mathcal{H} \wedge \left(\sum_\iota \geq \sum_i\right)$$

encode BT operation

enforce class c_ι instead of c_i

insane decimal point precision

MaxSAT

maximize $\sum_\iota - \sum_i$

subject to $\mathcal{H}$

Overview of the approach

SMT

- **reach logic, can handle *linear constraints***
- **directly reason about:**

$$\mathcal{H} \wedge \left(\sum_{\iota} \geq \sum_{i}\right)$$

encode BT operation

enforce class c_ι instead of c_i

insane decimal point precision

MaxSAT

maximize $\sum_{\iota} - \sum_{i}$

subject to $\mathcal{H}$

- **pure propositional logic**
  - prediction as objective function
  - weighted soft clauses keep precision
  - core-guided MaxSAT

Overview of the approach

SMT

- reach logic, can handle *linear constraints*
- directly reason about:

$$\mathcal{H} \wedge \left(\sum_\iota \geq \sum_i\right)$$

encode BT operation

enforce class c_ι instead of c_i

insane decimal point precision

MaxSAT

maximize $\sum_\iota - \sum_i$

subject to $\mathcal{H}$

- pure propositional logic
  - prediction as objective function
  - weighted soft clauses keep precision
  - core-guided MaxSAT
- incremental MaxSAT calls
  - *MiniSat-like assumptions interface!*
  - unsatisfiable core *reuse*

Overview of the approach

SMT

- reach logic, can handle *linear constraints*
- directly reason about:

$$\mathcal{H} \land \left(\sum_{\iota} \geq \sum_{i}\right)$$

encode BT operation

enforce class c_ι instead of c_i

insane decimal point precision

MaxSAT

maximize	$\sum_{\iota} - \sum_{i}$
subject to	$\mathcal{H}$

- pure propositional logic
  - prediction as objective function
  - weighted soft clauses keep precision
  - core-guided MaxSAT
- incremental MaxSAT calls
  - *MiniSat-like assumptions interface!*
  - unsatisfiable core *reuse*
- novel weight stratification

Overview of the approach

SMT

- reach logic, can handle *linear constraints*
- directly reason about:

$$\mathcal{H} \wedge \left(\sum_{\iota} \geq \sum_{i}\right)$$

encode BT operation

enforce class c_ι instead of c_i

insane decimal point precision

MaxSAT

maximize $\sum_{\iota} - \sum_{i}$

subject to $\mathcal{H}$

- pure propositional logic
  - prediction as objective function
  - weighted soft clauses keep precision
  - core-guided MaxSAT
- incremental MaxSAT calls
  - *MiniSat-like assumptions interface!*
  - unsatisfiable core *reuse*
- novel weight stratification
- early termination

Overview of the approach

SMT

- reach logic, can handle *linear constraints*
- directly reason about:

$$\mathcal{H} \wedge \left(\sum_{\iota} \geq \sum_{i}\right)$$

encode BT operation

enforce class c_ι instead of c_i

insane decimal point precision

MaxSAT

maximize $\sum_{\iota} - \sum_{i}$

subject to $\mathcal{H}$

- pure propositional logic
  - prediction as objective function
  - weighted soft clauses keep precision
  - core-guided MaxSAT
- incremental MaxSAT calls
  - *MiniSat-like assumptions interface!*
  - unsatisfiable core *reuse*
- novel weight stratification
- early termination

the devil is in the details

Encoding BT operation

$\mathcal{F}$ — **set of features** $\forall_{j \in \mathcal{F}}\ D_j$ — **domain of feature j** $\mathfrak{E}$ — **tree ensemble**

Encoding BT operation

$\mathcal{F}$ — set of features $\quad \forall_{j \in \mathcal{F}} \; D_j$ — domain of feature j $\quad \mathfrak{E}$ — tree ensemble

$$\mathcal{H} = \bigwedge_{j \in \mathcal{F}} \mathcal{H}_{D_j} \wedge \bigwedge_{T_i \in \mathfrak{E}} \mathcal{H}_{T_i}$$

Encoding BT operation

$\mathcal{F}$ — set of features $\quad \forall_{j \in \mathcal{F}}\ D_j$ — domain of feature j $\quad \mathfrak{E}$ — tree ensemble

$$\mathcal{H} = \bigwedge_{j \in \mathcal{F}} \mathcal{H}_{D_j} \wedge \bigwedge_{T_i \in \mathfrak{E}} \mathcal{H}_{T_i}$$

$\mathcal{H}_{D_j}$ encodes feature domain D_j

Encoding BT operation

$\mathcal{F}$ — set of features $\quad$ $\forall_{j \in \mathcal{F}} \; D_j$ — domain of feature j $\quad$ $\mathfrak{E}$ — tree ensemble

$$\mathcal{H} = \bigwedge_{j \in \mathcal{F}} \mathcal{H}_{D_j} \wedge \bigwedge_{T_i \in \mathfrak{E}} \mathcal{H}_{T_i}$$

$\mathcal{H}_{D_j}$ encodes feature domain D_j

- feature threshold values $s_{j,k}$ from $\mathfrak{E}$

Encoding BT operation

$\mathcal{F}$ — set of features $\quad \forall_{j \in \mathcal{F}}\ D_j$ — domain of feature j $\quad \mathfrak{E}$ — tree ensemble

$$\mathcal{H} = \bigwedge_{j \in \mathcal{F}} \mathcal{H}_{D_j} \wedge \bigwedge_{T_i \in \mathfrak{E}} \mathcal{H}_{T_i}$$

$\mathcal{H}_{D_j}$ encodes feature domain D_j

- feature threshold values $s_{j,k}$ from $\mathfrak{E}$
- variable $o_{j,k} = 1$ iff $x_j < s_{j,k}$
- variable $l_{j,k} = 1$ iff $x_j \in [s_{j,k-1}, s_{j,k})$

Encoding BT operation

$\mathcal{F}$ — set of features

$\forall_{j \in \mathcal{F}} \; D_j$ — domain of feature j

$\mathfrak{E}$ — tree ensemble

$$\mathcal{H} = \bigwedge_{j \in \mathcal{F}} \mathcal{H}_{D_j} \wedge \bigwedge_{T_i \in \mathfrak{E}} \mathcal{H}_{T_i}$$

$\mathcal{H}_{D_j}$ encodes feature domain D_j

- feature threshold values $s_{j,k}$ from $\mathfrak{E}$
- variable $o_{j,k} = 1$ iff $x_j < s_{j,k}$
- variable $l_{j,k} = 1$ iff $x_j \in [s_{j,k-1}, s_{j,k})$
- order encoding of D_j with $l_{j,k}$ and $o_{j,k}$
- instance is expressed by $l_{j,k}$-variables

Encoding BT operation

$\mathcal{F}$ — set of features $\quad \forall_{j\in\mathcal{F}}\ D_j$ — domain of feature j $\quad \mathfrak{E}$ — tree ensemble

$$\mathcal{H} = \bigwedge_{j\in\mathcal{F}} \mathcal{H}_{D_j} \wedge \bigwedge_{T_i\in\mathfrak{E}} \mathcal{H}_{T_i}$$

$\mathcal{H}_{D_j}$ encodes feature domain D_j

- feature threshold values $s_{j,k}$ from $\mathfrak{E}$
- variable $o_{j,k} = 1$ iff $x_j < s_{j,k}$
- variable $l_{j,k} = 1$ iff $x_j \in [s_{j,k-1}, s_{j,k})$
- order encoding of D_j with $l_{j,k}$ and $o_{j,k}$
- instance is expressed by $l_{j,k}$-variables

$\mathcal{H}_{T_i}$ encodes paths of T_i

Encoding BT operation

$\mathcal{F}$ — set of features $\forall_{j \in \mathcal{F}}$ D_j — domain of feature j $\mathfrak{E}$ — tree ensemble

$$\mathcal{H} = \bigwedge_{j \in \mathcal{F}} \mathcal{H}_{D_j} \wedge \bigwedge_{T_i \in \mathfrak{E}} \mathcal{H}_{T_i}$$

$\mathcal{H}_{D_j}$ encodes feature domain D_j

- feature threshold values $s_{j,k}$ from $\mathfrak{E}$
- variable $o_{j,k} = 1$ iff $x_j < s_{j,k}$
- variable $l_{j,k} = 1$ iff $x_j \in [s_{j,k-1}, s_{j,k})$
- order encoding of D_j with $l_{j,k}$ and $o_{j,k}$
- instance is expressed by $l_{j,k}$-variables

$\mathcal{H}_{T_i}$ encodes paths of T_i

- $\mathcal{P}_i$ is the set of paths of T_i
- given a $P_r \in \mathcal{P}_i$, t_r denotes its leaf

Encoding BT operation

$\mathcal{F}$ — set of features $\forall_{j\in\mathcal{F}}\ D_j$ — domain of feature j $\mathfrak{E}$ — tree ensemble

$$\mathcal{H} = \bigwedge_{j\in\mathcal{F}} \mathcal{H}_{D_j} \wedge \bigwedge_{T_i\in\mathfrak{E}} \mathcal{H}_{T_i}$$

$\mathcal{H}_{D_j}$ encodes feature domain D_j

- feature threshold values $s_{j,k}$ from $\mathfrak{E}$
- variable $o_{j,k} = 1$ iff $x_j < s_{j,k}$
- variable $l_{j,k} = 1$ iff $x_j \in [s_{j,k-1}, s_{j,k})$
- order encoding of D_j with $l_{j,k}$ and $o_{j,k}$
- instance is expressed by $l_{j,k}$-variables

$\mathcal{H}_{T_i}$ encodes paths of T_i

- $\mathcal{P}_i$ is the set of paths of T_i
- given a $P_r \in \mathcal{P}_i$, t_r denotes its leaf
- encode each path $P_r \in \mathcal{P}_i$:

$$\left(\bigwedge_{(x_j<s_{j,k})\in P_r} o_{j,k} \wedge \bigwedge_{(x_j\geq s_{j,k})\in P_r} \neg o_{j,k}\right) \leftrightarrow t_r$$

Encoding BT operation

$\mathcal{F}$ — set of features $\quad$ $\forall_{j \in \mathcal{F}}\ D_j$ — domain of feature j $\quad$ $\mathfrak{E}$ — tree ensemble

$$\mathcal{H} = \bigwedge_{j \in \mathcal{F}} \mathcal{H}_{D_j} \wedge \bigwedge_{T_i \in \mathfrak{E}} \mathcal{H}_{T_i}$$

$\mathcal{H}_{D_j}$ encodes feature domain D_j

- feature threshold values $s_{j,k}$ from $\mathfrak{E}$
- variable $o_{j,k} = 1$ iff $x_j < s_{j,k}$
- variable $l_{j,k} = 1$ iff $x_j \in [s_{j,k-1}, s_{j,k})$
- order encoding of D_j with $l_{j,k}$ and $o_{j,k}$
- instance is expressed by $l_{j,k}$-variables

$\mathcal{H}_{T_i}$ encodes paths of T_i

- $\mathcal{P}_i$ is the set of paths of T_i
- given a $P_r \in \mathcal{P}_i$, t_r denotes its leaf
- encode each path $P_r \in \mathcal{P}_i$:

$$\left(\bigwedge_{(x_j < s_{j,k}) \in P_r} o_{j,k} \wedge \bigwedge_{(x_j \geq s_{j,k}) \in P_r} \neg o_{j,k} \right) \leftrightarrow t_r$$

- $\sum_{P_r \in \mathcal{P}_i} t_r = 1$

Encoding BT operation

$\mathcal{F}$ — set of features $\forall_{j\in\mathcal{F}}$ D_j — domain of feature j $\mathfrak{E}$ — tree ensemble

$$\mathcal{H} = \bigwedge_{j\in\mathcal{F}} \mathcal{H}_{D_j} \wedge \bigwedge_{T_i\in\mathfrak{E}} \mathcal{H}_{T_i}$$

$\mathcal{H}_{D_j}$ encodes feature domain D_j

- feature threshold values $s_{j,k}$ from $\mathfrak{E}$
- variable $o_{j,k} = 1$ iff $x_j < s_{j,k}$
- variable $l_{j,k} = 1$ iff $x_j \in [s_{j,k-1}, s_{j,k})$
- order encoding of D_j with $l_{j,k}$ and $o_{j,k}$
- instance is expressed by $l_{j,k}$-variables

$\mathcal{H}_{T_i}$ encodes paths of T_i

- $\mathcal{P}_i$ is the set of paths of T_i
- given a $P_r \in \mathcal{P}_i$, t_r denotes its leaf
- encode each path $P_r \in \mathcal{P}_i$:

$$\left(\bigwedge_{(x_j<s_{j,k})\in P_r} o_{j,k} \wedge \bigwedge_{(x_j\geq s_{j,k})\in P_r} \neg o_{j,k}\right) \leftrightarrow t_r$$

- $\sum_{P_r\in\mathcal{P}_i} t_r = 1$

see paper for details

“Optimizing” model predictions

AXp condition for $\mathcal{X}$

$$\forall(\mathbf{x} \in \mathbb{F}).\bigwedge_{j\in\mathcal{X}}(x_j = v_j) \rightarrow (\boldsymbol{\tau}(\mathbf{x}) = \mathbf{c}_i)$$

"Optimizing" model predictions

AXp condition for $\mathcal{X}$

$$\forall(\mathbf{x} \in \mathbb{F}). \bigwedge_{j \in \mathcal{X}} (x_j = v_j) \rightarrow (\tau(\mathbf{x}) = \mathbf{c}_i)$$

$\mathcal{X}$ is not AXp

“Optimizing” model predictions

AXp condition for $\mathcal{X}$

$$\forall(\mathbf{x} \in \mathbb{F}). \bigwedge_{j\in\mathcal{X}}(x_j = v_j) \rightarrow (\tau(\mathbf{x}) = c_i)$$

$\mathcal{X}$ is not AXp **if** $\exists_{\mathbf{x}\in\mathbb{F}, c_\iota \neq c_i}. \ w_\iota(\mathbf{x}) \geq w_i(\mathbf{x})$

"Optimizing" model predictions

AXp condition for $\mathcal{X}$

$$\forall(\mathbf{x} \in \mathbb{F}). \bigwedge_{j \in \mathcal{X}}(x_j = v_j) \rightarrow (\tau(\mathbf{x}) = c_i)$$

weighted sums $\sum_\iota$ and $\sum_i$

$\mathcal{X}$ **is not AXp** **if** $\exists_{\mathbf{x} \in \mathbb{F}, c_\iota \neq c_i}. \; w_\iota(\mathbf{x}) \geq w_i(\mathbf{x})$

"Optimizing" model predictions

AXp condition for $\mathcal{X}$

$$\forall(\mathbf{x} \in \mathbb{F}).\bigwedge_{j\in\mathcal{X}}(x_j = v_j) \rightarrow (\tau(\mathbf{x}) = c_i)$$

$\mathcal{X}$ is not AXp if $\exists_{\mathbf{x}\in\mathbb{F},c_\iota\neq c_i}.\ w_\iota(\mathbf{x}) \geq w_i(\mathbf{x})$

weighted sums $\sum_\iota$ and $\sum_i$

hard to reason about

“Optimizing” model predictions

AXp condition for $\mathcal{X}$

$$\forall(\mathbf{x} \in \mathbb{F}). \bigwedge_{j\in\mathcal{X}}(x_j = v_j) \rightarrow (\tau(\mathbf{x}) = c_i)$$

weighted sums $\sum_\iota$ and $\sum_i$

$\mathcal{X}$ is not AXp if $\exists_{\mathbf{x}\in\mathbb{F}, c_\iota \neq c_i}. \; w_\iota(\mathbf{x}) \geq w_i(\mathbf{x})$

even to represent in propositional logic!

hard to reason about

"Optimizing" model predictions

AXp condition for $\mathcal{X}$

$$\forall(\mathbf{x} \in \mathbb{F}).\bigwedge_{j\in\mathcal{X}}(x_j = v_j) \rightarrow (\tau(\mathbf{x}) = c_i)$$

weighted sums $\sum_\iota$ and $\sum_i$

$\mathcal{X}$ is not AXp if $\exists_{\mathbf{x}\in\mathbb{F},c_\iota\neq c_i}.\ w_\iota(\mathbf{x}) \geq w_i(\mathbf{x})$

even to represent in propositional logic!

hard to reason about

maximize	$\mathcal{S}_{i,\iota} = \sum_\iota - \sum_i$
subject to	$\mathcal{H} \wedge \bigwedge_{j\in\mathcal{X}} \mathbb{l}[x_j = v_j]$

"Optimizing" model predictions

AXp condition for $\mathcal{X}$

$$\forall(\mathbf{x} \in \mathbb{F}). \bigwedge_{j\in\mathcal{X}}(x_j = v_j) \rightarrow (\tau(\mathbf{x}) = c_i)$$

weighted sums $\sum_\iota$ and $\sum_i$

$\mathcal{X}$ is not AXp if $\exists_{\mathbf{x}\in\mathbb{F}, c_\iota \neq c_i}. \; w_\iota(\mathbf{x}) \geq w_i(\mathbf{x})$

even to represent in propositional logic!

hard to reason about

$\mathcal{X}$ is not AXp if $\mathcal{S}^*_{i,\iota} \geqslant 0$

maximize $\mathcal{S}_{i,\iota} = \sum_\iota - \sum_i$

subject to $\mathcal{H} \wedge \bigwedge_{j\in\mathcal{X}} \mathbb{[}x_j = v_j\mathbb{]}$

"Optimizing" model predictions

AXp condition for $\mathcal{X}$

$$\forall(\mathbf{x} \in \mathbb{F}). \bigwedge_{j\in\mathcal{X}}(x_j = v_j) \rightarrow (\tau(\mathbf{x}) = c_i)$$

weighted sums $\sum_\iota$ and $\sum_i$

$\mathcal{X}$ is not AXp **if** $\exists_{\mathbf{x}\in\mathbb{F},c_\iota\neq c_i}.\ w_\iota(\mathbf{x}) \geq w_i(\mathbf{x})$

even to represent in propositional logic!

hard to reason about

$\mathcal{X}$ is not AXp if $\mathcal{S}^*_{i,\iota} \geqslant 0$

maximize	$\mathcal{S}_{i,\iota} = \sum_\iota - \sum_i$
subject to	$\mathcal{H} \wedge \bigwedge_{j\in\mathcal{X}} \mathbb{I}[x_j = v_j]$

weighted soft clauses + propositional variables only

multiple calls to a MaxSAT solver with varying assumptions $\mathcal{X}$

multiple calls to a MaxSAT solver
with varying assumptions $\mathcal{X}$

the need for incremental MaxSAT!

Incrementality in MaxSAT

Function MaxSAT(ϕ, $\mathcal{A}$):

Input: ϕ: Partial CNF formula ($\phi \triangleq \mathcal{H} \wedge \mathcal{S}$)

$\mathcal{A}$: Set of assumption literals

Output: μ: MaxSAT model

```
cost ← 0                                  # initially, cost is 0

𝒞 ← ValidCores(ϕ, 𝒜)                      # get valid unsatisfiable cores

foreach κ ∈ 𝒞:                            # iterate over known cores κ
    cost ← cost + CoreWt(κ)               # add its weight to cost

    ϕ ← Process(ϕ, κ)                     # process κ and update ϕ

while SAT(ϕ, 𝒜) = false:                  # iterate until ϕ gets satisfiable
    κ ← GetCore(ϕ)                        # new unsatisfiable core

    cost ← cost + CoreWt(κ)               # add its weight to cost

    ϕ ← Process(ϕ, κ)                     # process κ and update ϕ

    Record(ϕ, 𝒜, κ)                       # record κ for the future

return GetModel(ϕ)                        # ϕ is now satisfiable
```

Incrementality in MaxSAT

Function MaxSAT(ϕ, $\mathcal{A}$):

Input: ϕ: Partial CNF formula ($\phi \triangleq \mathcal{H} \wedge \mathcal{S}$)

$\mathcal{A}$: Set of assumption literals

Output: μ: MaxSAT model

cost $\leftarrow 0$ *# initially, cost is 0*

$\mathcal{C} \leftarrow$ ValidCores($\phi, \mathcal{A}$) *# get valid unsatisfiable cores*

foreach $\kappa \in \mathcal{C}$: *# iterate over known cores* κ

  cost $\leftarrow$ *cost* + CoreWt(κ) *# add its weight to cost*

  $\phi \leftarrow$ Process(ϕ, κ) *# process* κ *and update* ϕ

while SAT(ϕ, $\mathcal{A}$) = false: *# iterate until* ϕ *gets satisfiable*

  $\kappa \leftarrow$ GetCore(ϕ) *# new unsatisfiable core*

  cost $\leftarrow$ *cost* + CoreWt(κ) *# add its weight to cost*

  $\phi \leftarrow$ Process(ϕ, κ) *# process* κ *and update* ϕ

  Record($\phi, \mathcal{A}, \kappa$) *# record* κ *for the future*

return GetModel(ϕ) *#* ϕ *is now satisfiable*

MiniSat-like assumptions interface:

- assumptions $\mathcal{A} \triangleq \{ l[\![x_j = v_j]\!] \mid j \in \mathcal{X} \}$

Incrementality in MaxSAT

Function MAXSAT(ϕ, $\mathcal{A}$):

Input: ϕ: Partial CNF formula ($\phi \triangleq \mathcal{H} \wedge \mathcal{S}$)

$\mathcal{A}$: Set of assumption literals

Output: μ: MaxSAT model

cost $\leftarrow 0$ *# initially, cost is 0*

$\mathcal{C} \leftarrow$ VALIDCORES(ϕ, $\mathcal{A}$) *# get valid unsatisfiable cores*

foreach $\kappa \in \mathcal{C}$: *# iterate over known cores κ*

assumptions are passed down to SAT solver

$\phi \leftarrow$ PROCESS(ϕ, κ) *# process κ and update ϕ*

while SAT(ϕ, $\mathcal{A}$) $=$ false: *# iterate until ϕ gets satisfiable*

$\kappa \leftarrow$ GETCORE(ϕ) *# new unsatisfiable core*

cost $\leftarrow$ *cost* $+$ COREWT(κ) *# add its weight to cost*

$\phi \leftarrow$ PROCESS(ϕ, κ) *# process κ and update ϕ*

RECORD(ϕ, $\mathcal{A}$, κ) *# record κ for the future*

return GETMODEL(ϕ) *# ϕ is now satisfiable*

MiniSat-like assumptions interface:

- assumptions $\mathcal{A} \triangleq \{ l[\![x_j = v_j]\!] \mid j \in \mathcal{X} \}$

Incrementality in MaxSAT

Function MaxSAT$(\phi, \mathcal{A})$:

Input: ϕ: Partial CNF formula ($\phi \triangleq \mathcal{H} \wedge \mathcal{S}$)

$\mathcal{A}$: Set of assumption literals

Output: μ: MaxSAT model

cost $\leftarrow 0$ *# initially, cost is 0*

$\mathcal{C} \leftarrow$ ValidCores$(\phi, \mathcal{A})$ *# get valid unsatisfiable cores*

foreach $\kappa \in \mathcal{C}$: *# iterate over known cores* κ

assumptions are passed down to SAT solver

$\phi \leftarrow$ Process(ϕ, κ) *# process κ and update ϕ*

while SAT$(\phi, \mathcal{A}) =$ false: *# iterate until ϕ gets satisfiable*

$\kappa \leftarrow$ GetCore(ϕ) *# new unsatisfiable core*

cost $\leftarrow$ *cost* $+$ CoreWt(κ) *# add its weight to cost*

$\phi \leftarrow$ Process(ϕ, κ) *# process κ and update ϕ*

Record$(\phi, \mathcal{A}, \kappa)$ *# record κ for the future*

return GetModel(ϕ) *# ϕ is now satisfiable*

MiniSat-like assumptions interface:

- assumptions $\mathcal{A} \triangleq \{l[\![x_j = \nu_j]\!] \mid j \in \mathcal{X}\}$
- can get *"unsat core"* $\mathcal{A}* = \cup \mathcal{A}'$ s.t. $\mathcal{A}' = \kappa \cap \mathcal{A}$

Incrementality in MaxSAT

Function MaxSAT$(\phi, \mathcal{A})$:

Input: ϕ: Partial CNF formula ($\phi \triangleq \mathcal{H} \wedge \mathcal{S}$)

$\mathcal{A}$: Set of assumption literals

Output: μ: MaxSAT model

cost $\leftarrow 0$ — *# initially, cost is 0*

$\mathcal{C} \leftarrow$ ValidCores$(\phi, \mathcal{A})$ — *# get valid unsatisfiable cores*

foreach $\kappa \in \mathcal{C}$: — *# iterate over known cores κ*

assumptions are passed down to SAT solver

$\phi \leftarrow$ Process(ϕ, κ) — *# process κ and update ϕ*

while SAT$(\phi, \mathcal{A}) =$ false: — *# iterate until ϕ gets satisfiable*

$\kappa \leftarrow$ GetCore(ϕ) — *# new unsatisfiable core*

cost $\leftarrow$ *cost* $+$ CoreWt(κ) — *# add its weight to cost*

$\phi \leftarrow$ Process(ϕ, κ) — *# process κ and update ϕ*

Record$(\phi, \mathcal{A}, \kappa)$ — *# record κ for the future*

return GetModel(ϕ) — *# ϕ is now satisfiable*

MiniSat-like assumptions interface:

- assumptions $\mathcal{A} \triangleq \{l[\![x_j = v_j]\!] \mid j \in \mathcal{X}\}$
- can get *"unsat core"* $\mathcal{A}* = \cup \mathcal{A}'$ s.t. $\mathcal{A}' = \kappa \cap \mathcal{A}$
- *similar to modern SAT solvers*

Incrementality in MaxSAT

Function $\textsc{MaxSAT}(\phi, \mathcal{A})$:

Input: ϕ: Partial CNF formula ($\phi \triangleq \mathcal{H} \wedge \mathcal{S}$)

$\mathcal{A}$: Set of assumption literals

Output: μ: MaxSAT model

```
cost ← 0                                   # initially, cost is 0
𝒞 ← ValidCores(ϕ, 𝒜)                       # get valid unsatisfiable cores
foreach κ ∈ 𝒞:                             # iterate over known cores κ
    cost ← cost + CoreWt(κ)                # add its weight to cost
    ϕ ← Process(ϕ, κ)                      # process κ and update ϕ
while SAT(ϕ, 𝒜) = false:                   # iterate until ϕ gets satisfiable
    κ ← GetCore(ϕ)                         # new unsatisfiable core
    cost ← cost + CoreWt(κ)                # add its weight to cost
    ϕ ← Process(ϕ, κ)                      # process κ and update ϕ
    Record(ϕ, 𝒜, κ)                        # record κ for the future
return GetModel(ϕ)                         # ϕ is now satisfiable
```

MiniSat-like assumptions interface:

- assumptions $\mathcal{A} \triangleq \{l[\![x_j = v_j]\!] \mid j \in \mathcal{X}\}$
- can get *"unsat core"* $\mathcal{A}* = \cup \mathcal{A}'$ s.t. $\mathcal{A}' = \kappa \cap \mathcal{A}$
- *similar to modern SAT solvers*

unsatisfiable core reuse:

Incrementality in MaxSAT

Function MaxSAT(ϕ, $\mathcal{A}$):

Input: ϕ: Partial CNF formula ($\phi \triangleq \mathcal{H} \wedge \mathcal{S}$)
$\mathcal{A}$: Set of assumption literals

Output: μ: MaxSAT model

```
cost ← 0                                   # initially, cost is 0
𝒞 ← ValidCores(ϕ, 𝒜)                        # get valid unsatisfiable cores
foreach κ ∈ 𝒞:                              # iterate over known cores κ
    cost ← cost + CoreWt(κ)                 # add its weight to cost
    ϕ ← Process(ϕ, κ)                       # process κ and update ϕ
while SAT(ϕ, 𝒜) = false:                    # iterate until ϕ gets satisfiable
    κ ← GetCore(ϕ)                          # new unsatisfiable core
    cost ← cost + CoreWt(κ)                 # add its weight to cost
    ϕ ← Process(ϕ, κ)                       # process κ and update ϕ
    Record(ϕ, 𝒜, κ)                         # record κ for the future
return GetModel(ϕ)                          # ϕ is now satisfiable
```

MiniSat-like assumptions interface:

- assumptions $\mathcal{A} \triangleq \{l[\![x_j = v_j]\!] \mid j \in \mathcal{X}\}$
- can get *"unsat core"* $\mathcal{A}* = \cup \mathcal{A}'$ s.t. $\mathcal{A}' = \kappa \cap \mathcal{A}$
- *similar to modern SAT solvers*

unsatisfiable core reuse:

- record each new core κ_i

Incrementality in MaxSAT

Function $\textsc{MaxSAT}(\phi, \mathcal{A})$:

Input: ϕ: Partial CNF formula ($\phi \triangleq \mathcal{H} \wedge \mathcal{S}$)

$\mathcal{A}$: Set of assumption literals

Output: μ: MaxSAT model

cost $\leftarrow 0$ *# initially, cost is 0*

$\mathcal{C} \leftarrow \textsc{ValidCores}(\phi, \mathcal{A})$ *# get valid unsatisfiable cores*

foreach $\kappa \in \mathcal{C}$**:** *# iterate over known cores* κ

  cost $\leftarrow$ *cost* $+ \textsc{CoreWt}(\kappa)$ *# add its weight to cost*

  $\phi \leftarrow \textsc{Process}(\phi, \kappa)$ *# process* κ *and update* ϕ

while $\text{SAT}(\phi, \mathcal{A}) = \text{false}$**:** *# iterate until* ϕ *gets satisfiable*

  $\kappa \leftarrow \textsc{GetCore}(\phi)$ *# new unsatisfiable core*

  cost $\leftarrow$ *cost* $+ \textsc{CoreWt}(\kappa)$ *# add its weight to cost*

  $\phi \leftarrow \textsc{Process}(\phi, \kappa)$ *# process* κ *and update* ϕ

  $\textsc{Record}(\phi, \mathcal{A}, \kappa)$ *# record* κ *for the future*

return $\textsc{GetModel}(\phi)$ *#* ϕ *is now satisfiable*

MiniSat-like assumptions interface:

- assumptions $\mathcal{A} \triangleq \{l[\![x_j = v_j]\!] \mid j \in \mathcal{X}\}$
- can get *"unsat core"* $\mathcal{A}* = \cup \mathcal{A}'$ s.t. $\mathcal{A}' = \kappa \cap \mathcal{A}$
- *similar to modern SAT solvers*

unsatisfiable core reuse:

- record each new core κ_i
- start a new call by **reusing known cores**

Incrementality in MaxSAT

Function MAXSAT($\phi, \mathcal{A}$):

Input: ϕ: Partial CNF formula ($\phi \triangleq \mathcal{H} \wedge \mathcal{S}$)

$\mathcal{A}$: Set of assumption literals

Output: μ: MaxSAT model

cost $\leftarrow 0$ *# initially, cost is 0*

$\mathcal{C} \leftarrow$ VALIDCORES($\phi, \mathcal{A}$) *# get valid unsatisfiable cores*

foreach $\kappa \in \mathcal{C}$: *# iterate over known cores κ*

  cost $\leftarrow$ *cost* + COREWT(κ) *# add its weight to cost*

  $\phi \leftarrow$ PROCESS(ϕ, κ) *# process κ and update ϕ*

while SAT(ϕ, $\mathcal{A}$) = false: *# iterate until ϕ gets satisfiable*

  $\kappa \leftarrow$ GETCORE(ϕ) *# new unsatisfiable core*

  cost $\leftarrow$ *cost* + COREWT(κ) *# add its weight to cost*

  $\phi \leftarrow$ PROCESS(ϕ, κ) *# process κ and update ϕ*

  RECORD($\phi, \mathcal{A}, \kappa$) *# record κ for the future*

return GETMODEL(ϕ) *# ϕ is now satisfiable*

MiniSat-like assumptions interface:

- assumptions $\mathcal{A} \triangleq \{l[\![x_j = v_j]\!] \mid j \in \mathcal{X}\}$
- can get *"unsat core"* $\mathcal{A}* = \cup\mathcal{A}'$ s.t. $\mathcal{A}' = \kappa \cap \mathcal{A}$
- *similar to modern SAT solvers*

unsatisfiable core reuse:

- record each new core κ_i
  - use external SAT solver
  - record all the literals core κ_i *depends on*
  - i.e. use $\bigwedge_{l \in \mathcal{D}_i} \rightarrow \zeta_i$
- start a new call by **reusing known cores**

Incrementality in MaxSAT

Function MaxSAT($\phi, \mathcal{A}$):

Input: ϕ: Partial CNF formula ($\phi \triangleq \mathcal{H} \wedge \mathcal{S}$)

$\mathcal{A}$: Set of assumption literals

Output: μ: MaxSAT model

cost $\leftarrow 0$ *# initially, cost is 0*

$\mathcal{C} \leftarrow$ ValidCores$(\phi, \mathcal{A})$ *# get valid unsatisfiable cores*

foreach $\kappa \in \mathcal{C}$: *# iterate over known cores* κ

  cost $\leftarrow$ *cost* + CoreWt(κ) *# add its weight to cost*

  $\phi \leftarrow$ Process(ϕ, κ) *# process* κ *and update* ϕ

while SAT$(\phi, \mathcal{A})$ = false: *# iterate until* ϕ *gets satisfiable*

  $\kappa \leftarrow$ GetCore(ϕ) *# new unsatisfiable core*

  cost $\leftarrow$ *cost* + CoreWt(κ) *# add its weight to cost*

  $\phi \leftarrow$ Process(ϕ, κ) *# process* κ *and update* ϕ

  Record$(\phi, \mathcal{A}, \kappa)$ *# record* κ *for the future*

return GetModel(ϕ) *#* ϕ *is now satisfiable*

MiniSat-like assumptions interface:

- assumptions $\mathcal{A} \triangleq \{l[\![x_j = v_j]\!] \mid j \in \mathcal{X}\}$
- can get *"unsat core"* $\mathcal{A}* = \cup\mathcal{A}'$ s.t. $\mathcal{A}' = \kappa \cap \mathcal{A}$
- *similar to modern SAT solvers*

unsatisfiable core reuse:

- record each new core κ_i
  - use external SAT solver
  - record all the literals core κ_i *depends on*
  - i.e. use $\bigwedge_{l \in \mathcal{D}_i} \to \zeta_i$
- start a new call by **reusing known cores**
  - apply unit propagation given $\mathcal{S}$ and $\mathcal{A}$
  - all reusable cores are *"propagated"* in the right order

Incrementality in MaxSAT

Function MaxSAT$(\phi, \mathcal{A})$:

Input: ϕ: Partial CNF formula ($\phi \triangleq \mathcal{H} \wedge \mathcal{S}$)
$\mathcal{A}$: Set of assumption literals

Output: μ: MaxSAT model

cost $\leftarrow 0$ *# initially, cost is 0*

$\mathcal{C} \leftarrow$ ValidCores$(\phi, \mathcal{A})$ *# get valid unsatisfiable cores*

foreach $\kappa \in \mathcal{C}$: *# iterate over known cores κ*

  cost $\leftarrow$ *cost* + CoreWt(κ) *# add its weight to cost*

  $\phi \leftarrow$ Process(ϕ, κ) *# process κ and update ϕ*

while SAT$(\phi, \mathcal{A}) =$ false: *# iterate until ϕ gets satisfiable*

  $\kappa \leftarrow$ GetCore(ϕ) *# new unsatisfiable core*

  cost $\leftarrow$ *cost* + CoreWt(κ) *# add its weight to cost*

  $\phi \leftarrow$ Process(ϕ, κ) *# process κ and update ϕ*

  Record$(\phi, \mathcal{A}, \kappa)$ *# record κ for the future*

return GetModel(ϕ) *# ϕ is now satisfiable*

MiniSat-like assumptions interface:

- assumptions $\mathcal{A} \triangleq \{l[\![x_j = v_j]\!] \mid j \in \mathcal{X}\}$
- can get *"unsat core"* $\mathcal{A}* = \cup\mathcal{A}'$ s.t. $\mathcal{A}' = \kappa \cap \mathcal{A}$
- *similar to modern SAT solvers*

unsatisfiable core reuse:

- record each new core κ_i
  - use external SAT solver
  - record all the literals core κ_i *depends on*
  - i.e. use $\bigwedge_{l \in \mathcal{D}_i} \to \zeta_i$
- start a new call by **reusing known cores**
  - apply unit propagation given $\mathcal{S}$ and $\mathcal{A}$
  - all reusable cores are *"propagated"* in the right order

early termination:

- no need to solve the problem to optimality!

Incrementality in MaxSAT

Function MaxSAT(ϕ, $\mathcal{A}$):

Input: ϕ: Partial CNF formula ($\phi \triangleq \mathcal{H} \wedge \mathcal{S}$)

$\mathcal{A}$: Set of assumption literals

Output: μ: MaxSAT model

cost $\leftarrow 0$ *# initially, cost is 0*

$\mathcal{C} \leftarrow$ ValidCores($\phi, \mathcal{A}$) *# get valid unsatisfiable cores*

foreach $\kappa \in \mathcal{C}$: *# iterate over known cores* κ

    cost $\leftarrow$ *cost* $+$ CoreWt(κ) *# add its weight to cost*

    $\phi \leftarrow$ Process(ϕ, κ) *# process* κ *and update* ϕ

while SAT(ϕ, $\mathcal{A}$) $=$ false**:** *# iterate until* ϕ *gets satisfiable*

    $\kappa \leftarrow$ GetCore(ϕ) *# new unsatisfiable core*

    cost $\leftarrow$ *cost* $+$ CoreWt(κ) *# add its weight to cost*

LB on the cost is updated at each iteration

return GetModel(ϕ) *#* ϕ *is now satisfiable*

MiniSat-like assumptions interface:

- assumptions $\mathcal{A} \triangleq \{l[\![x_j = v_j]\!] \mid j \in \mathcal{X}\}$
- can get *"unsat core"* $\mathcal{A}* = \cup \mathcal{A}'$ s.t. $\mathcal{A}' = \kappa \cap \mathcal{A}$
- *similar to modern SAT solvers*

unsatisfiable core reuse:

- record each new core κ_i
    - use external SAT solver
    - record all the literals core κ_i *depends on*
    - i.e. use $\bigwedge_{l \in \mathcal{D}_i} \rightarrow \zeta_i$
- start a new call by **reusing known cores**
    - apply unit propagation given $\mathcal{S}$ and $\mathcal{A}$
    - all reusable cores are *"propagated"* in the right order

early termination:

- no need to solve the problem to optimality!
- no misclassification if *cost* $> \sum_\iota$

Incrementality in MaxSAT

Function MAXSAT(ϕ, $\mathcal{A}$):

Input: ϕ: Partial CNF formula ($\phi \triangleq \mathcal{H} \wedge \mathcal{S}$)

$\mathcal{A}$: Set of assumption literals

Output: μ: MaxSAT model

```
cost ← 0                                  # initially, cost is 0
C ← VALIDCORES(φ, A)                      # get valid unsatisfiable cores
foreach κ ∈ C:                            # iterate over known cores κ
    cost ← cost + COREWT(κ)               # add its weight to cost
    φ ← PROCESS(φ, κ)                     # process κ and update φ
while SAT(φ, A) = false:                  # iterate until φ gets satisfiable
    κ ← GETCORE(φ)                        # new unsatisfiable core
    cost ← cost + COREWT(κ)               # add its weight to cost
    φ ← PROCESS(φ, κ)                     # process κ and update φ
    RECORD(φ, A, κ)                       # record κ for the future
return GETMODEL(φ)                        # φ is now satisfiable
```

can over-approximate objective once current level is solved

MiniSat-like assumptions interface:

- assumptions $\mathcal{A} \triangleq \{l[\![x_j = v_j]\!] \mid j \in \mathcal{X}\}$
- can get *"unsat core"* $\mathcal{A}* = \cup \mathcal{A}'$ s.t. $\mathcal{A}' = \kappa \cap \mathcal{A}$
- *similar to modern SAT solvers*

unsatisfiable core reuse:

- record each new core κ_i
  - use external SAT solver
  - record all the literals core κ_i *depends on*
  - i.e. use $\bigwedge_{l \in \mathcal{D}_i} \rightarrow \zeta_i$
- start a new call by **reusing known cores**
  - apply unit propagation given $\mathcal{S}$ and $\mathcal{A}$
  - all reusable cores are *"propagated"* in the right order

early termination:

- no need to solve the problem to optimality!
- no misclassification if *cost* $> \sum_\iota$
- misclassification if over-approximation's objective > 0

Additional heuristic — distance-based stratification

why?

why? — many weighted clauses with unique weights

BLO and diversity-based stratification fail to apply!

why? — many weighted clauses with unique weights

BLO and diversity-based stratification fail to apply!

$$\left| w - \frac{\sum_{(c_i, w_i) \in \mathcal{L}} w_i}{|\mathcal{L}|} \right| < \left| w - \frac{\sum_{(c_j, w_j) \in \mathcal{S} \wedge w_j < w} w_j}{|\{(c_j, w_j) \in \mathcal{S} \wedge w_j < w\}|} \right|$$

why? — many weighted clauses with unique weights

BLO and diversity-based stratification fail to apply!

$$\left| w - \frac{\sum_{(c_i, w_i) \in \mathcal{L}} w_i}{|\mathcal{L}|} \right| < \left| w - \frac{\sum_{(c_j, w_j) \in \mathcal{S} \wedge w_j < w} w_j}{|\{(c_j, w_j) \in \mathcal{S} \wedge w_j < w\}|} \right|$$

$(\neg t_1, 0.72284)$ $(t_2, 0.41527)$ $(\neg t_3, 0.41645)$ $(t_4, 0.16249)$ $(t_5, 0.72452)$

why? — many weighted clauses with unique weights

BLO and diversity-based stratification fail to apply!

$$\left| w - \frac{\sum_{(c_i, w_i) \in \mathcal{L}} w_i}{|\mathcal{L}|} \right| < \left| w - \frac{\sum_{(c_j, w_j) \in \mathcal{S} \wedge w_j < w} w_j}{|\{(c_j, w_j) \in \mathcal{S} \wedge w_j < w\}|} \right|$$

$(\neg t_1, 0.72284)$ $(t_2, 0.41527)$ $(\neg t_3, 0.41645)$ $(t_4, 0.16249)$ $(t_5, 0.72452)$

Additional heuristic — distance-based stratification

why? — many weighted clauses with unique weights

BLO and diversity-based stratification fail to apply!

$$\left| w - \frac{\sum_{(c_i, w_i) \in \mathcal{L}} w_i}{|\mathcal{L}|} \right| < \left| w - \frac{\sum_{(c_j, w_j) \in \mathcal{S} \land w_j < w} w_j}{|\{(c_j, w_j) \in \mathcal{S} \land w_j < w\}|} \right|$$

$(\neg t_1, 0.72284)$ $(t_2, 0.41527)$ $(\neg t_3, 0.41645)$ $(t_4, 0.16249)$ $(t_5, 0.72452)$

$$\mathcal{L}_1 = \{ (t_5, 0.72452) \}$$

why? — many weighted clauses with **unique weights**

BLO and diversity-based stratification fail to apply!

$$\left| w - \frac{\sum_{(c_i,w_i)\in\mathcal{L}} w_i}{|\mathcal{L}|} \right| < \left| w - \frac{\sum_{(c_j,w_j)\in\mathcal{S}\wedge w_j<w} w_j}{|\{(c_j,w_j)\in\mathcal{S}\wedge w_j<w\}|} \right|$$

$(\neg t_1, 0.72284)$ $(t_2, 0.41527)$ $(\neg t_3, 0.41645)$ $(t_4, 0.16249)$ $(t_5, 0.72452)$

$$0.00168 = |0.72284 - 0.72452| < |0.72284 - {}^{(0.41645+0.41527+0.16249)}/_3| = 0.39144$$

$$\mathcal{L}_1 = \left\{ (t_5, 0.72452) \right\}$$

why? — many weighted clauses with unique weights

BLO and diversity-based stratification fail to apply!

$$\left| w - \frac{\sum_{(c_i, w_i) \in \mathcal{L}} w_i}{|\mathcal{L}|} \right| < \left| w - \frac{\sum_{(c_j, w_j) \in \mathcal{S} \wedge w_j < w} w_j}{|\{(c_j, w_j) \in \mathcal{S} \wedge w_j < w\}|} \right|$$

$(\neg t_1, 0.72284)$ $(t_2, 0.41527)$ $(\neg t_3, 0.41645)$ $(t_4, 0.16249)$ $(t_5, 0.72452)$

$$\mathcal{L}_1 = \left\{ \begin{array}{c} (t_5, 0.72452) \\ (\neg t_1, 0.72284) \end{array} \right\}$$

why? — many weighted clauses with **unique weights**

BLO and diversity-based stratification fail to apply!

$$\left| w - \frac{\sum_{(c_i,w_i)\in\mathcal{L}} w_i}{|\mathcal{L}|} \right| < \left| w - \frac{\sum_{(c_j,w_j)\in\mathcal{S}\wedge w_j<w} w_j}{|\{(c_j,w_j)\in\mathcal{S}\wedge w_j<w\}|} \right|$$

$(\neg t_1, 0.72284)$ $(t_2, 0.41527)$ $(\neg t_3, 0.41645)$ $(t_4, 0.16249)$ $(t_5, 0.72452)$

$0.30723 = |0.41645 - {}^{0.72452+0.72284}/_2| > |0.41645 - {}^{(0.41527+0.16249)}/_2| = 0.12757$

$$\mathcal{L}_1 = \left\{ \begin{array}{c} (t_5, 0.72452) \\ (\neg t_1, 0.72284) \end{array} \right\}$$

Additional heuristic — distance-based stratification

why? — many weighted clauses with unique weights

BLO and diversity-based stratification fail to apply!

$$\left| w - \frac{\sum_{(c_i, w_i) \in \mathcal{L}} w_i}{|\mathcal{L}|} \right| < \left| w - \frac{\sum_{(c_j, w_j) \in \mathcal{S} \wedge w_j < w} w_j}{|\{(c_j, w_j) \in \mathcal{S} \wedge w_j < w\}|} \right|$$

$(\neg t_1, 0.72284)$ $(t_2, 0.41527)$ $(\neg t_3, 0.41645)$ $(t_4, 0.16249)$ $(t_5, 0.72452)$

$$\mathcal{L}_1 = \left\{ \begin{array}{c} (t_5, 0.72452) \\ (\neg t_1, 0.72284) \end{array} \right\} \qquad \mathcal{L}_2 = \left\{ \begin{array}{c} (\neg t_3, 0.41645) \\ \end{array} \right\}$$

why? — many weighted clauses with unique weights

BLO and diversity-based stratification fail to apply!

$$\left| w - \frac{\sum_{(c_i,w_i)\in\mathcal{L}} w_i}{|\mathcal{L}|} \right| < \left| w - \frac{\sum_{(c_j,w_j)\in\mathcal{S}\wedge w_j<w} w_j}{|\{(c_j,w_j)\in\mathcal{S}\wedge w_j<w\}|} \right|$$

$(\neg t_1, 0.72284)$ $(t_2, 0.41527)$ $(\neg t_3, 0.41645)$ $(t_4, 0.16249)$ $(t_5, 0.72452)$

$0.00118 = |0.41527 - 0.41645| < |0.41527 - 0.16249| = 0.25278$

$$\mathcal{L}_1 = \left\{ \begin{array}{c} (t_5, 0.72452) \\ (\neg t_1, 0.72284) \end{array} \right\} \qquad \mathcal{L}_2 = \left\{ \begin{array}{c} (\neg t_3, 0.41645) \\ \end{array} \right\}$$

why? — many weighted clauses with **unique weights**

BLO and diversity-based stratification fail to apply!

$$\left| w - \frac{\sum_{(c_i,w_i)\in\mathcal{L}} w_i}{|\mathcal{L}|} \right| < \left| w - \frac{\sum_{(c_j,w_j)\in\mathcal{S}\wedge w_j<w} w_j}{|\{(c_j,w_j)\in\mathcal{S}\wedge w_j<w\}|} \right|$$

$(\neg t_1, 0.72284)$ $(t_2, 0.41527)$ $(\neg t_3, 0.41645)$ $(t_4, 0.16249)$ $(t_5, 0.72452)$

$$\mathcal{L}_1 = \left\{ \begin{array}{c} (t_5, 0.72452) \\ (\neg t_1, 0.72284) \end{array} \right\} \qquad \mathcal{L}_2 = \left\{ \begin{array}{c} (\neg t_3, 0.41645) \\ (t_2, 0.41527) \end{array} \right\}$$

why? — many weighted clauses with **unique weights**

BLO and diversity-based stratification fail to apply!

$$\left| w - \frac{\sum_{(c_i, w_i) \in \mathcal{L}} w_i}{|\mathcal{L}|} \right| < \left| w - \frac{\sum_{(c_j, w_j) \in \mathcal{S} \wedge w_j < w} w_j}{|\{(c_j, w_j) \in \mathcal{S} \wedge w_j < w\}|} \right|$$

$(\neg t_1, 0.72284)$ $(t_2, 0.41527)$ $(\neg t_3, 0.41645)$ $(t_4, 0.16249)$ $(t_5, 0.72452)$

$0.25337 = |0.16249 - {}^{0.41527+0.41645}/_2| > |0.16249 - 0| = 0.16249$

$$\mathcal{L}_1 = \left\{ \begin{array}{c} (t_5, 0.72452) \\ (\neg t_1, 0.72284) \end{array} \right\} \qquad \mathcal{L}_2 = \left\{ \begin{array}{c} (\neg t_3, 0.41645) \\ (t_2, 0.41527) \end{array} \right\}$$

Additional heuristic — distance-based stratification

why? — many weighted clauses with unique weights

BLO and diversity-based stratification fail to apply!

$$\left| w - \frac{\sum_{(c_i, w_i) \in \mathcal{L}} w_i}{|\mathcal{L}|} \right| < \left| w - \frac{\sum_{(c_j, w_j) \in \mathcal{S} \wedge w_j < w} w_j}{|\{(c_j, w_j) \in \mathcal{S} \wedge w_j < w\}|} \right|$$

$(\neg t_1, 0.72284)$ $(t_2, 0.41527)$ $(\neg t_3, 0.41645)$ $(t_4, 0.16249)$ $(t_5, 0.72452)$

$$\mathcal{L}_1 = \left\{ \begin{array}{c} (t_5, 0.72452) \\ (\neg t_1, 0.72284) \end{array} \right\} \qquad \mathcal{L}_2 = \left\{ \begin{array}{c} (\neg t_3, 0.41645) \\ (t_2, 0.41527) \end{array} \right\} \qquad \mathcal{L}_3 = \{(t_4, 0.16249)\}$$

Experimental results

Experimental setup

- **machine configuration:**
  - Intel Core i5 2.30GHz with 8GByte RAM

Experimental setup

- **machine configuration:**
  - Intel Core i5 2.30GHz with 8GByte RAM
  - running macOS Big Sur

Experimental setup

- **machine configuration:**
  - Intel Core i5 2.30GHz with 8GByte RAM
  - running macOS Big Sur
  - no time or memory limit

Experimental setup

- **machine configuration:**
  - Intel Core i5 2.30GHz with 8GByte RAM
  - running macOS Big Sur
  - no time or memory limit
- **UCI Machine Learning Repository** + **Penn Machine Learning Benchmarks**

Experimental setup

- **machine configuration:**
  - Intel Core i5 2.30GHz with 8GByte RAM
  - running macOS Big Sur
  - no time or memory limit
- **UCI Machine Learning Repository + Penn Machine Learning Benchmarks**
  - **21 datasets**
    - 7–60 features, 26.95 on average
    - 59–58000 data instances, 4684.47 on average

Experimental setup

- **machine configuration:**
  - Intel Core i5 2.30GHz with 8GByte RAM
  - running macOS Big Sur
  - no time or memory limit
- **UCI Machine Learning Repository** + **Penn Machine Learning Benchmarks**
  - **21 datasets**
    - 7–60 features, 26.95 on average
    - 59–58000 data instances, 4684.47 on average
  - **XGBoost models**
    - depth 3–4
    - 50 trees per class

Experimental setup

- **machine configuration:**
  - Intel Core i5 2.30GHz with 8GByte RAM
  - running macOS Big Sur
  - no time or memory limit
- **UCI Machine Learning Repository + Penn Machine Learning Benchmarks**
  - **21 datasets**
    - 7–60 features, 26.95 on average
    - 59–58000 data instances, 4684.47 on average
  - **XGBoost models**
    - depth 3–4
    - 50 trees per class
  - **200 randomly selected** instances to explain
    - 3665 individual benchmarks in total

Experimental setup

- **competition tested:**
  - **SMT-based formal explainer**
    - makes calls to **Z3 through PySMT**
    - same explanation quality

[1] `https://github.com/alexeyignatiev/xreason/`

Experimental setup

- **competition tested:**
  - **SMT-based formal explainer**
    - makes calls to **Z3 through PySMT**
    - same explanation quality
  - **Anchor heuristic explainer**
    - sampling-based
    - *no* explanation quality guarantees
    - all parameters are set to default values

[1] `https://github.com/alexeyignatiev/xreason/`

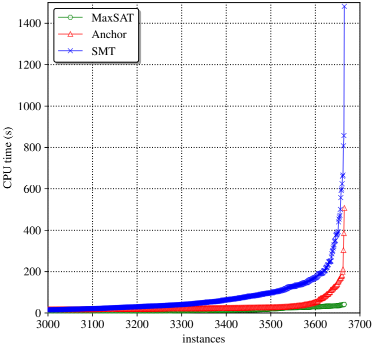
Experimental setup

- **competition tested:**
  - SMT-based formal explainer
    - makes calls to Z3 through PySMT
    - same explanation quality
  - Anchor heuristic explainer
    - sampling-based
    - *no* explanation quality guarantees
    - all parameters are set to default values

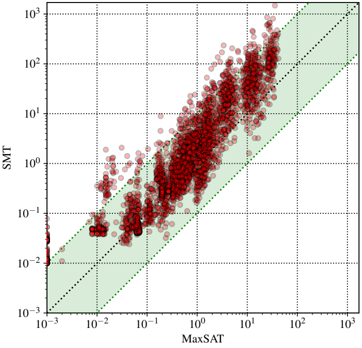
- **prototype**[1]
  - MaxSAT-based
    - **same code base** as in the SMT-based variant
    - applies **incrementally extended** **RC2**
    - makes calls to Glucose 3 through PySAT

[1] https://github.com/alexeyignatiev/xreason/

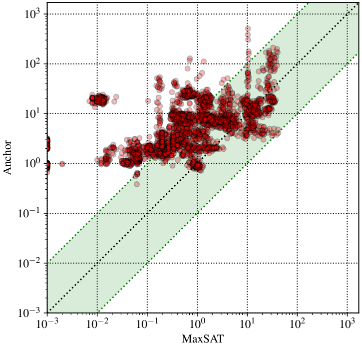
Experimental results



all three competitors

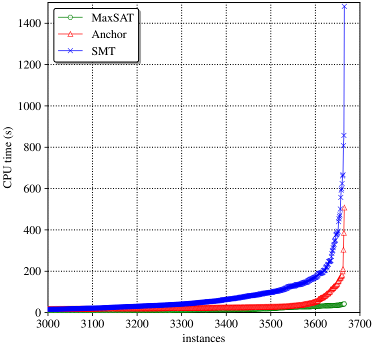


MaxSAT vs. SMT

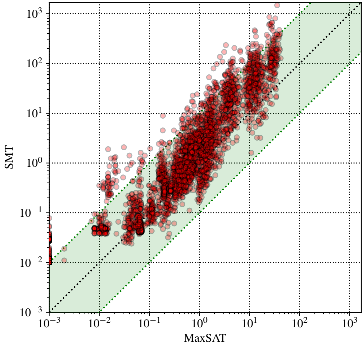


MaxSAT vs. Anchor

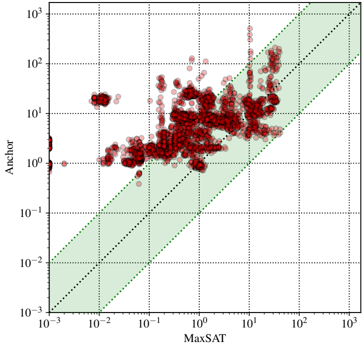
Experimental results



all three competitors



MaxSAT vs. SMT



MaxSAT vs. Anchor

up to 2 orders of magnitude performance improvement

more robust than SMT and Anchor

(see paper for details)

Summary and future work

Summary and future work

- **novel approach to explaining boosted trees**
  - **pure propositional logic**
    - prediction as objective function
    - core-guided MaxSAT

Summary and future work

- **novel approach to explaining boosted trees**
  - **pure propositional logic**
    - prediction as objective function
    - core-guided MaxSAT
  - **incremental MaxSAT calls**
    - *MiniSat-like assumptions interface!*
    - unsatisfiable core *reuse*

Summary and future work

- **novel approach to explaining boosted trees**
  - **pure propositional logic**
    - prediction as objective function
    - core-guided MaxSAT
  - **incremental MaxSAT calls**
    - *MiniSat-like assumptions interface!*
    - unsatisfiable core *reuse*
  - **novel weight stratification**
  - **early termination**

Summary and future work

- **novel approach to explaining boosted trees**
  - **pure propositional logic**
    - prediction as objective function
    - core-guided MaxSAT
  - **incremental MaxSAT calls**
    - *MiniSat-like assumptions interface!*
    - unsatisfiable core *reuse*
  - **novel weight stratification**
  - **early termination**
  - **a few orders of magnitude** performance improvement

Summary and future work

- **novel approach to explaining boosted trees**
  - **pure propositional logic**
    - prediction as objective function
    - core-guided MaxSAT
  - **incremental MaxSAT calls**
    - *MiniSat-like assumptions interface!*
    - unsatisfiable core *reuse*
  - **novel weight stratification**
  - **early termination**
  - **a few orders of magnitude** performance improvement
- **future work**
  - **contrastive explanations with MaxSAT**

Summary and future work

- **novel approach to explaining boosted trees**
  - **pure propositional logic**
    - prediction as objective function
    - core-guided MaxSAT
  - **incremental MaxSAT calls**
    - *MiniSat-like assumptions interface!*
    - unsatisfiable core *reuse*
  - **novel weight stratification**
  - **early termination**
  - **a few orders of magnitude** performance improvement
- **future work**
  - **contrastive explanations with MaxSAT**
  - **same ideas for other ML models?**

Summary and future work

- **novel approach to explaining boosted trees**
  - **pure propositional logic**
    - prediction as objective function
    - core-guided MaxSAT
  - **incremental MaxSAT calls**
    - *MiniSat-like assumptions interface!*
    - unsatisfiable core *reuse*
  - **novel weight stratification**
  - **early termination**
  - **a few orders of magnitude** performance improvement
- **future work**
  - **contrastive explanations with MaxSAT**
  - **same ideas for other ML models?**
  - **more applications of incremetal MaxSAT?**

Questions?